Toward General Mathematical Game Playing Agents

Daniel Ashlock, Eun-Youn Kim, and Diego Perez-Liebana

***Abstract*—General game playing AI and general video game playing AI are both active research areas. Mathematical games, like prisoner's dilemma, rock-paper-scissors, or the snowdrift game can also be played by general purpose agents. An agent representation in which agents play knowing only their own score and their opponent's score in a previous round is used to enable general mathematical game playing. Agents are trained to recognize when the game being played has changed and trained to play competently on diverse sets of games. This domain, simpler than general game playing or general video game playing, is offered as a test lab for concepts in the creation of general AI. It is demonstrated that agents can be trained to play three different coordination games competently. The agents also learn to play prisoner's dilemma and a coordination game that reverses which moves lead to high payoffs.**

I. Introduction

General game playing AIs seek to imitate the human ability to learn and then play, at least competently, any of a broad variety of games. Any person that plays games knows that skill levels of their opponents not only vary, but vary between games. This incredibly broad mission poses a remarkably hard problem for AI training. General video game playing is a similar endeavor with a more restricted domain [20], [2]. This yielded a test domain which is more reasonable but still quite broad. Current research is driven by contests in which agents are asked to learn games they have not encountered before.

In this study it is proposed to examine *general mathematical game playing* (GMGP), an even more restricted domain. Mathematical games include games like the iterated prisoner's dilemma [14], the snowdrift game [13], and rock-paper-scissors [3]. This study will focus on simultaneous two-player games which are defined by a payoff matrix that specifies which moves are available to the players and the payoff to each player for each pair of moves.

II. Background

Games are commonly used as an approximation to General Purpose AI, a field in AI in which agents must be able to respond well to multiple and unprecedented situations. They present a multitude of different scenarios, in a single or multi-player mode, in which simulations and repetitions are generally fast and available. They also capture multiple facets of character behaviour, such as hidden information, bluffing and opponent modeling, plus the complexities of random processes, as the ones determined by shuffling cards or rolling dies.

Not surprisingly, the literature shows a recent proliferation on frameworks for general game (GGP) and video game playing (GVGP). Examples are the GGP benchmark [16], the Arcade Learning Environment (ALE; [7]), OpenAI Gym [8] or the General Video Game AI (GVGAI) framework [19]. All these frameworks propose the creation of agents that should be able to play any game is given to them, often from different perspectives: single or multi player, board or video-games, via screen capture or object-based state information, etc. For this paper, we focus on the GVGAI framework, both due to its recent popularity and 2-player track.

GVGAI is a framework written in Java that proposes an interface for agents to play any of its 160 games (at the time of writing). Concretely, there are 100 single player and 60 two-player games, which have been developed over the years following its main associated competitions [20], [10]. These games are written in the Video Game Description Language (VGDL), a text based language that is able to describe two-dimensional grid arcade video games, of the like of Space Invaders, Pacman or Sokoban.

VGDL was initially proposed by Ebner et al. [9] at the Dagstuhl Seminar on Artificial and Computational Intelligence in Games in 2013, and then developed further by Tom Schaul [21] in a python-based framework for agent planning and learning. Games are defined using two different files: a game and a level description. While the latter specifies which sprites go where in a two-dimensional grid at the beginning of the game, the former defines four different constituents: the set of sprites (plus parameters) that take part in the game, the rules that govern their interactions, the conditions upon which the game ends and a mapping between characters in the level file and the sprites allowed.

Agents (also referred to as *controllers*) in GVGAI do not have access to the VGDL description of the games, but they receive (in the single and two-player planning settings) information about the current game state in two different methods that need to be implemented. The first one is a constructor, which can be used to initialize any data structure needed by the controllers, and the second one is an *act* method that is called once every frame, which requests an action to be executed by the player (or avatar) at that time step. Given that GVGAI operates in real-time, these methods must return before 1 second and 40 milliseconds respectively

Daniel Ashlock is with the Department of Mathematics and Statistics at the University of Guelph, in Guelph, Ontario, Canada, N1G 2W1, dashlock@uoguelph.ca

Eun-Youn Kim is at School of Basic Sciences, Hanbat National University, 125 Dongseodae-ro, Yuseong-gu, Daejeon 34158, Republic of Korea, eunykim00@gmail.com

Diego Perez-Lebana is with the Game AI Research Group at the Queen Mary University of London, Mile End Rd, London E1 4NS, UK, dperez@essex.ac.uk

The authors thank the University of Guelph and Canadian Natural Sciences and Engineering Research Council of Canada (NSERC) for supporting this work.

to avoid any penalties.

The information agents receive on each one of these methods not only contains details about the current game state (i.e. current game tick, score, avatar features and presence of other sprites), but also provides a Forward Model (FM) for controllers to roll the current state forward upon the supply of potential actions that can be returned. This FM has been used by multiple approaches in the literature [19] to develop agents based on simulation techniques such as Monte Carlo Tree Search (MCTS; [18]) or Rolling Horizon Evolutionary Algorithms [11].

For the two-player case, games use simultaneous moves, therefore the FM requires that the agent also supplies the action that (supposedly) the opponent would take. This requires, in this particular case, an effort from the agent to model the behaviour of the other player, which is still an interesting and open area of research. In their work, J. Gonzalez-Castro and Diego Perez-Leiebana [12] studied 9 different opponent models for an MCTS controller, from simple ones (assuming the opponent will use the same / the opposite action used in the last tick, or that it will use the action with the best / worst / average estimated reward) to more complex ones (using models computed over the distribution of actions used by an MCTS agent both on multiple games and during the last n frames in the current game). The authors showed that these latter approaches where able to outperform a random opponent model, which was the main model used by all top players of the competition [10].

However, further research is needed in the topic of adapting to the other player in the game. For instance, similar to the games proposed in this paper, a 20% of the 2-player games existing in the GVGAI framework are cooperative, and agents are not given the information regarding if they must compete to win or collaborate in order to achieve a common goal. Additionally, to the knowledge of the authors, no research has been undertaken in GVGAI to adapt and learn from the opponent's behaviour during the course of several repetitions of the game being played. Although the learning track of the competition [15] features agent learning via playing repetitions (with no FM), at the moment this setting is restricted to single-player games only.

The present paper attempts not only to extend the kind of games GVGAI agents can play, but also to give some initial steps in two-player opponent modeling and adaptation for general video-game playing. The agents in this study learn if they are supposed to cooperate or compete via evolutionary learning; this at least established context for alternate player modeling methods, implicitly embedded in the agent's state space by evolution.

III. Design of Experiments

A population of agents are taught to play multiple games simultaneously. This is accomplished by changing which game is being played during fitness evaluation. Agents trained on two games would play a round-robin tournament with 150 iterations of the first game, then 150 more of the second. The goal is to have the agents evolve to notice that the game has changed based only on the payoff they are receiving. The need to act based only on payoff information informs the design of the agent representation.

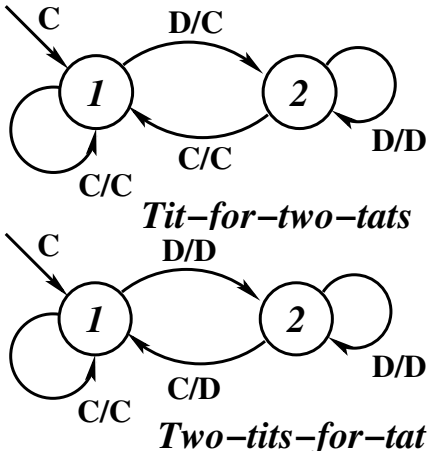

Fig. 1. Finite state agents that encode two well-know strategies for the iterated prisoner's dilemma.

Figure 1 shows two examples of finite state agents that play iterated prisoner's dilemma. These are Mealy-style machines whose transitions are driven by the opponent's last action: cooperation or defection. These agents must know that there are two moves and what those moves are. An agent trained for GMGP will not have knowledge in advance of the number or identity of moves in the game.

This design issue is addressed in this study by creating agents that based their actions on the score that they and their opponent received on the last round. Some knowledge is still required: a range in which the scores of all the games will fall.

A. The Agent Representation

The agent representation used is both relatively novel and being used in a new setting. It is a new form of, *binary decision automata* (BDA). This representation was first introduced in [17] to model stress in the workplace and used to explore the effect of changing information resources available to game playing agents in [6]. An example of one of the BDA used in this study is given in Figure 2. These agents are Mealy architecture finite state machines. Transitions are driven by a Boolean test, in this case based on a threshold value stored in each state and the scores obtained by the agent and its opponent in previous rounds. Three tests are available: is my score larger than the threshold, is my opponent's score larger than the threshold, and is my opponent's score larger than my own. Transitions are based on true and false outcomes of these tests. The novelty of the representation is that it has no access to knowledge about what moves are being made, only what payoffs it receives. This means that the agent can detect and adapt to situations in which the rules are changed, something demonstrated in the experimental section.

State	Test	If True	If False
0:	If (His>My)	C→0;	C→1;
1:	If (My>0.27)	A→3;	A→0;
2:	If (His>My)	A→2;	A→3;
3:	If (His>3.48)	B→3;	B→0;

Fig. 2. A 4-state binary decision automata, evolved to play a game with three moves **A**, **B**, and **C**.

The agent is stored as a vector of states. The action taken for the true transition of the zeroth state is the agent's initial action, used to begin iterated play. During reproduction, two point crossover on the vector of states is used, followed by 1-MNM mutations with the number of mutations selected uniformly at random; MNM is the *maximum number of mutations*, a parameter of the representation. A mutation picks a state uniformly at random and then generate a new value for the state's threshold or one of its two transitions and actions.

In both initialization and mutation, a *shape* [5] is used that forbids any state to make a transition to itself. Preliminary work while debugging the agent software demonstrated that self-looping states that play a single move form a large local optima in the agent space. Blocking this possibility improves performance and encourages the agents to learn more complicated strategies that at least have the potential to detect and respond to a change of game.

B. The Evolutionary Algorithm

The evolutionary algorithm maintains a population of 72 agents, a number selected by preliminary experimentation to supply sufficient population diversity to learn multiple games. The agents are initialized by filling in the threshold values, actions, and transitions uniformly at random. Threshold values are selected uniformly at random in the range from the lowest to highest score an agent can receive. The agents are evolved for 250 generations, sufficient time to allow the population to stabilize genetically after burning away the agent's initial randomness.

Fitness evaluation is a round-robin iterated tournament on all pairs of distinct agents. Iterated play is continued for a number of rounds specified for each experiment. The game being played is changed during fitness evaluation so that the agent's score is the average of the scores obtained while playing multiple different games. This is one of several possible ways to expose an agent to multiple games; it was chosen because it is one of the simplest. This issue is discussed in Section V.

Selection and replacement are performed as follows. Agents are sorted into fitness order and an elite of the 48 most fit agents (two-thirds) are preserved. The twelve most fit pairs of agents are copied in fitness order. Succesive pairs of copies undergo crossover and mutation to produce 24 new agents, replacing the worst 24 in the population.

The retention of two-thirds of the agents, the *elite fraction* is a fairly important parameter. There is a substantial change in the genetic stability of an evolving population of game playing agents when this fraction is one-half or less [1]. In this study agents are given *at least* 100 rounds of play with each game with the goal of proof-of-concept that the agents can be trained to play multiple mathematical games. That informs the choice of a relatively conservative and high elite fraction of two-thirds.

C. Games Used

Experiments begin with parameter setting on sets of two and three simple coordination games. Figure 3 gives the payoff matrix for the three-move coordination games used. The first game has $X = 6$, $Y = 3$ and $Z = 1$; the second $X = 3$, $Y = 1$ and $Z = 6$; the third $X = 1$, $Y = 6$ and $Z = 3$. All of the games give a positive payoff for any coordination. The high, low, and middle payoff values are rotated to create games with substantially different payoff schemes.

Coordination Games

	A	B	C
A	X	0	0
B	0	Y	0
C	0	0	Z

Fig. 3. These matrices give the score a player making the move indexing the row of the matrix will receive against an opponent who makes the move indexing the column.

Prisoner's Dilemma or Coordination

Prisoner's Dilemma

	M_1	M_2
M_1	3	0
M_2	5	1

Coordination

	M_1	M_2
M_1	1	0
M_2	0	3

Fig. 4. These matrices give two games. The first is prisoner's dilemma, the second is a coordination game that rewards coordination on the move that corresponds to defection in the prisoner's more highly.

Two parameter studies are performed. The first uses the first two coordination games, the second uses all three coordination games. These games have three Nash equilibria, represented by the pure strategies. In the course of the fitness evaluation, changing which game is being played changes which of the Nash equilibria have the highest, middle, and lowest payoff.

The best of the tested parameters (eight states, $MNM = 1$) is then used for an experiment with the pair of games given in Figure 4. Here fitness is evaluated on the iterated prisoner's dilemma for 150 rounds, then on a coordination game that has positive payoffs only for coordination for an additional 150 rounds. To make the problem challenging the better coordination payoff, three, is for the move that corresponds to defection in Prisoner's dilemma, while coordination on the move corresponding to cooperation in the prisoner's dilemma pays one.

D. Analysis Tools

During the course of play, the agent's average and maximum score in each generation are recorded in each generation. In the final generation, the number of each of the possible pairs of plays is recorded and the fraction of each type calculated. This creates a matrix of the following form.

$$\begin{bmatrix} 0.440 & 0.013 & 0.000 \\ 0.012 & 0.325 & 0.000 \\ 0.000 & 0.000 & 0.210 \end{bmatrix}$$

This matrix is drawn from one of the runs that achieved a high score on three games. Any off-diagonal play represents moves that gained the players no score. A good agent will place most of its moves in the diagonal entries of this matrix. In the experiments with two games, we should see similar numbers in positions (1,1) and (3,3) because these represent moves scoring the maximum possible value of 6 at some point. When three games are played we should see roughly equal distribution along the diagonal in a player that has learned to shift among the games properly.

Two statistics are extracted to check the agent's behavior. The first is the sum of the diagonal of the matrix; this is the fraction of coordinated moves or *coordination fraction*.

The second statistic uses the *Shannon entropy* of the normalized diagonal entries to estimate the evenness of distribution of the coordinated moves. This entropy is maximized by an even division among diagonal entries; the normalized diagonal will yield a score near one if divided evenly between two possibilities and 1.59 if evenly divided between three. These numbers, 1.0 and 1.59 indicate correctly learning to shift between games in agents that also have a high coordination score for the two-game and three-game coordination experiments, respectively.

E. Runs performed

A parameter study was performed for agents with 4, 6, and 8 states using mutation rates $MNM \in \{1, 3, 5\}$. These nine sets of parameters were studied for fitness evaluation with two and three coordination games. The round robin tournaments for fitness evaluation were run for 300 rounds of play. These were 150 rounds each for two games or 100 each for three games, with the games always presented in the same order and played for contiguous blocks of time.

IV. Results and Discussion

Figure 5 shows the three types of outcomes that occurred when three games were used. Unlike prisoner's dilemma, the agents in the coordination game cannot exploit one another. The only options are coordination or failure to coordinate. The costs and benefits of each of these outcomes is the same for both players. The sole issue here is the difficulty of learning to detect when the game has shifted.

The top panel in Figure 5 shows a population of agents that first discover the "just play one move" strategy. The use of the shape forbidding self-loops in the machine enables the sort of escape this population makes from that local optima to a better local optima where the payoffs are 6, 6, and 3 in the different phases of the game and then, at about generation 150, to adopt a strategy that gets a payoff of 6 in all three phases of the game. There is some inefficiency in the shift between games, and the population average score is reduced by the presence of mutant strategies, but these agents have discovered a general strategy for the three games and learned to detect a shift between the games.

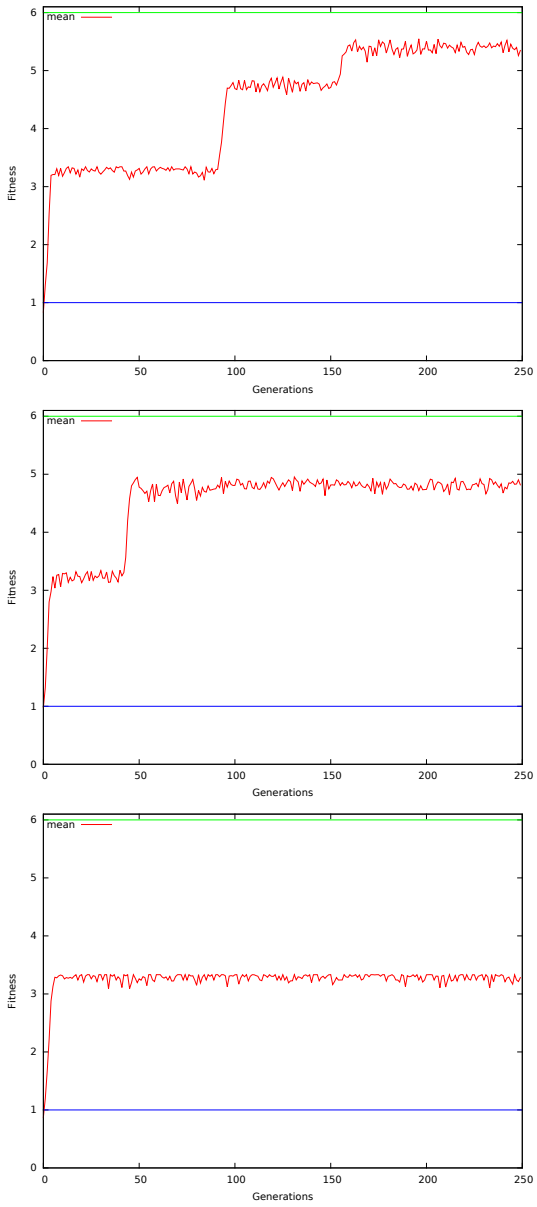

Fig. 5. Fitness over the course of evolution for three exemplary population from the runs using three games with 8 states and $MNM = 1$. The top run shows a population of agents that learned to play all three games; the middle track learned two games and accepted the second-best payoff for the third; the bottom track fails to shift games efficiently, playing a single move to get 6, 3, and 1 during different phases of fitness evaluation. The blue and green lines show the maximum and minimum values the average score can attain.

The second panel of Figure 5 shows the first escape to an optima what gets payoffs of 6, 6, and 3, in some order, in the different phases of the game. This population never makes

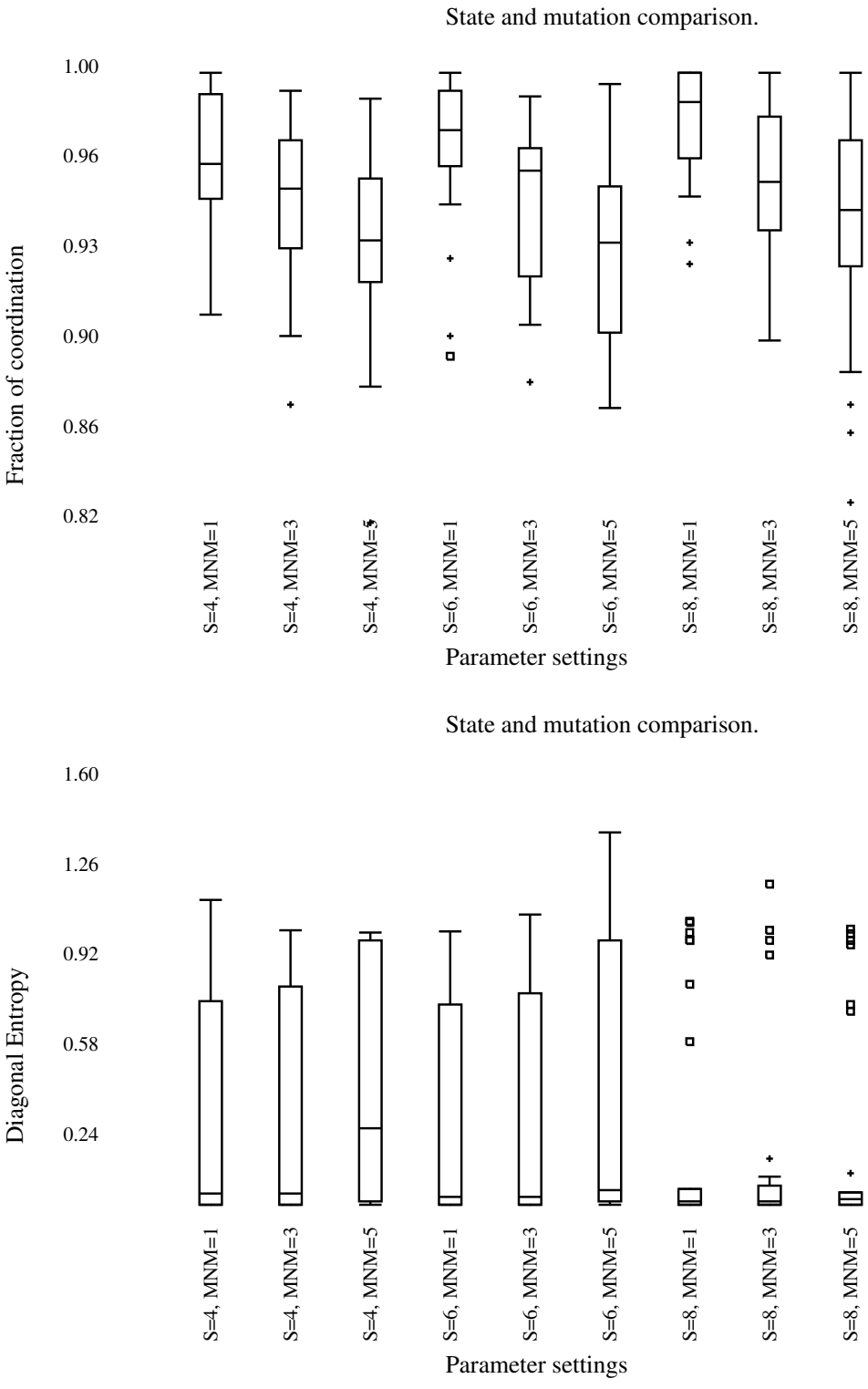

Fig. 6. Shown are the distribution of fraction of coordinated plays (upper panel) and the diagonal play entropy (lower panel) of populations of machines trained on play of two different coordination games.

the escape to the final global optima. The third panel – one of the most common outcomes – shows agents that stay in the "play one move" local optima throughout evolution.

These exemplary runs were taken from the best performing parameters set, with the most states and the lowest mutation rate. This suggests that increasing the mutation rate makes discovery of "just play one move" strategies that use two or more states (recall one state versions are forbidden by shaping). This suggests that more complex measures might be needed to prevent this optima from being the most common outcome, but the shuffled coordination games are simple a demonstration environment in any case. This problem may not exist in more complex environments.

Figure 6 shows the fraction-of-coordination results for the two game experiments while Figure 7 shows the same quantities for the three game experiments. These results suggest more states and lower mutation rates are good. The entropy results suggest that correct shifting among the games occurred but was rare for both the two and three game experiments.

Figure 8 shows the outcomes of the experiments with a version of prisoner's dilemma and a coordination game that reverses prisoner's dilemma cooperate-defect advantage structure. In thirty runs, twenty-four adopted "always move two" which corresponds to the always defect Nash equilibrium of the prisoner's dilemma and obtains the payoff of 3 from the coordination game. Three runs adopted mixed strategies. Three of the runs managed to detect the change of games and correctly shift between them.

When playing just iterated prisoner's dilemma, there is a tendency for fitness to rise and fall during the course of evolution as strategies drift away from their defensive ability though lack of use, followed by periods of exploitation. This was not visible in any of the thirty runs performed in the PD/Coordination experiments. This suggests that the need to deal with the coordination game stabilizes the agents in strategy space.

The plots in Figure 8 display the population mean and average fitness. This was done because exploitation in prisoner's dilemma corresponds to divergence of the average and maximum score. Very little such divergence occurred, even in the three runs with relatively complex mixed strategies.

V. Conclusions and Next Steps

This study presents a new representation for agents that play mathematical games and demonstrate that it can learn to play games based only on cues from scores obtained.

A. Better Training for General Agents

The fitness used to train agents for multiple games was a simple summing of the results of playing all the games involved. The games were intentionally scaled to have similar ranges of payoffs; if they had not been the lower payoff games might have been ignored. The sum-the-scores strategy makes the three problems decomposable. The agents can solve each problem individually.

In fact a few agent populations did learn to decompose the two or three games in all the collections of games used, but in all cases this was a small minority outcome. This means we gain proof-of-concept for general mathematical game playing but with a large dose of future work in improving the training algorithm.

One natural choice is multi-criteria optimization in which the score in each game is treated as a separate coordinate. This is a far more difficult training strategy, but one that seems likely to generate true general competitors across the training set.

A less complex solution to the problem consists of adding games one at a time. Permit the agent population to learn a first game, then add another game to the menu. This sort of serial addition of games may make the decomposition of the problem into distinct games more evolution-friendly.

B. Deep Time and True Generality

This study shows that the BDA-representation for agents can produce agents that recognize and correctly play games. They do not do so all that reliably and, this is more problematic, are only shown to be general across the games that serve as their training examples. A truly general mathematical game playing agent should be able to adapt to a game it has never seen before. This is clearly a domain for future work.

Hope appears in the way that agents, even when one of the games was the iterated prisoner's dilemma, tended to exhibit monotone increase in fitness. Figure 9 shows the way fitness evolves when two populations of agents are playing iterated prisoner's dilemma against one another. The monotonicity of fitness suggests that serial game addition may work.

Another factor is that the number of generations, 250, used in all the simulations was chosen based on earlier work in the iterated prisoner's dilemma. Evolving for 250 generations is more than enough to generate complex behavior. In [4] new prisoner's dilemma strategies were found to arise after 32768 (2^15) generations of evolution. It may be that better performance will result from simply running evolution for a longer time.

References

[1] Ashlock and E. Y. Kim. The impact of elite fraction and population size on evolved iterated prisoner's dilemma agents. In *Proceedings of the IEEE 2016 Congress on Evolutionary Computation*, pages 364–371, Piscataway NJ, 2016. IEEE Press.

[2] D. Ashlock, E. Y. Kim, and D. P. Lebana. General video game playing escapes the no free lunch theorem. In *Proceedings of the 2017 IEEE Conference on Computational Intelligence in Games*, pages 1–8, Piscataway NJ, 2017. IEEE Press.

[3] D. Ashlock and J. Schonfeld. Deriving card games from mathematical games. *Game and Puzzle Design*, 1(2):55–63, 2015.

[4] W. Ashlock and D. Ashlock. Changes in prisoner's dilemma strategies over evolutionary time with different population sizes. In *Proceedings of the 2006 Congress On Evolutionary Computation*, pages 1001–1008, Piscataway, NJ, 2006. IEEE press.

[5] W. Ashlock and D. Ashlock. Shaped prisoner's dilemma automata. In *Proceedings of the IEEE Conference on Computatiational Inteligence in Games*, pages 76–83, 2014.

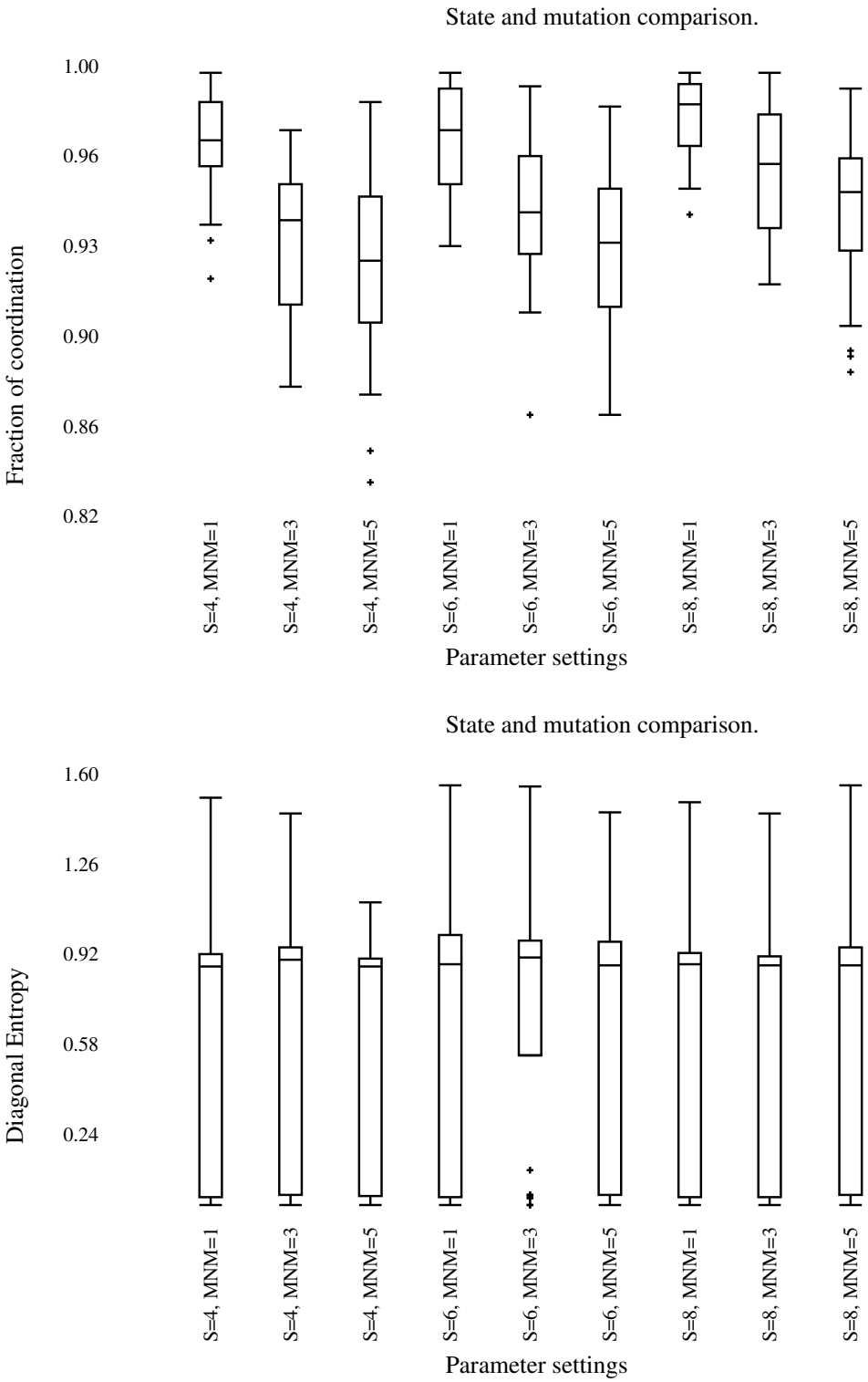

Fig. 7. Shown are the distribution of fraction of coordinated plays (upper panel) and the diagonal play entropy (lower panel) of populations of machines trained on play of three different different coordination games.

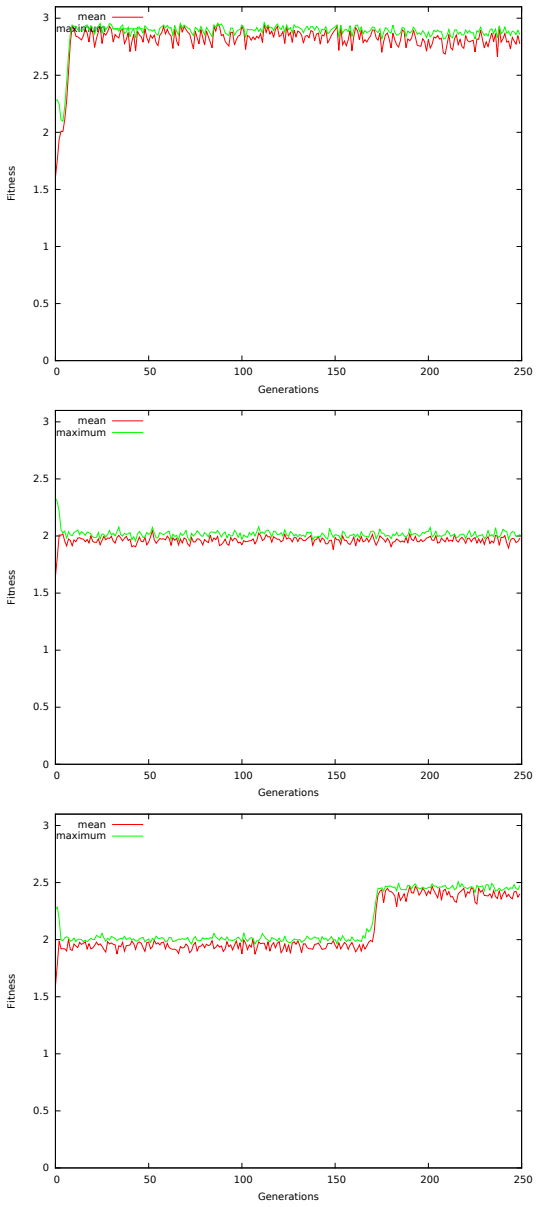

Fig. 8. Fitness over the course of evolution for three exemplary population from the runs using prisoner's dilemma and a coordination game with 8 states and $MNM = 1$. The top run shows a population of agents that learned to correctly shift between games; the middle track learned to shift games only some of the time, alternating moves in a locally optimal fashion; the bottom track fails to shift games efficiently, playing a single move to get 3 and 1 during different phases of training.

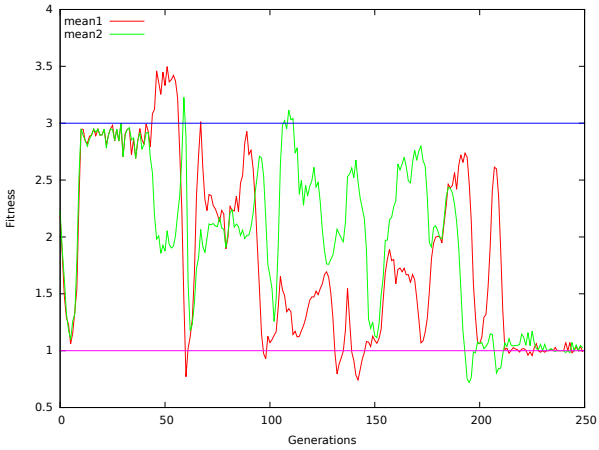

Fig. 9. An example of the non-monotone change of fitness when training agents to play prisoner's dilemma. This plot shows the average fitness of two competing populations; the blue and violet lines show the bounds that a population not engaging in active exploitation must stay between.

[6] L. Barlow and D. Ashlock. Varying decision inputs in prisoner's dilemma. In *Proceedings of the 2015 IEEE Conference on Computational Intelligence in Bioioinformatics and Comutational Biology*, pages 1–8, 2015.

[7] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[9] Marc Ebner, John Levine, Simon M Lucas, Tom Schaul, Tommy Thompson, and Julian Togelius. Towards a video game description language. *Dagstuhl Follow-Ups*, 6, 2013.

[10] Raluca D. Gaina, Adrien Couëtoux, Dennis J.N.J. Soemers, Mark H.M. Winands, Tom Vodopivec, Florian Kirchgessner, Jialin Liu, Simon M. Lucas, and Diego Perez-Liebana. The 2016 two-player gvgai competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 99:1–11, 2017.

[11] Raluca D Gaina, Simon M Lucas, and Diego Pérez-Liébana. Rolling Horizon Evolution Enhancements in General Video Game Playing. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2017.

[12] Jose M Gonzalez-Castro and Diego Perez-Liebana. Opponent Models Comparison for 2 Players in GVGAI Competitions. In *Computer Science and Electronic Engineering Conference (CEEC), 2017 9th*, pages 1–6. IEEE, 2017.

[13] G.W. Greenwood. Enhanced cooperation in the n-person iterated snowdrift game through tag mediation. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 1–8, Aug 2011.

[14] J. Li, P. Hingston, and G. Kendall. Engineering design of strategies for winning iterated prisoner's dilemma competitions. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(4):348–360, Dec 2011.

[15] Jialin Liu, Diego Perez-Liebana, and Simon M. Lucas. The Single-Player GVGAI Learning Framework - Technical Manual. Technical report, Queen Mary University of London, 2017.

[16] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General Game Playing: Game Description Language Specification. Technical report, Stanford Logic Group Computer Science Department Stanford University, Technical Report LG-2006-01, 2008.

[17] M. Page and D. Ashlock. Stress and productivity performance in the workforce modelled with binary decision automata. In *Proceedings of the 2015 IEEE Conference on Computational Intelligence in Bioioinformatics and Comutational Biology*, pages 1–8, 2015.

[18] Diego Perez, Spyridon Samothrakis, and Simon Lucas. Knowledge-based fast evolutionary mcts for general video game playing. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014.

[19] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Simon M Lucas, and Tom Schaul. General Video Game AI: Competition, Challenges and Opportunities. In *Thirtieth AAAI Conference on Artificial Intelligence*, pages 4335–4337, 2016.

[20] Diego Perez Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M Lucas, Adrien Couëtoux, Jerry Lee, Chong-U Lim, and Tommy Thompson. The 2014 General Video Game Playing Competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):229–243, 2016.

[21] Tom Schaul. A Video Game Description Language for Model-based or Interactive Learning. In *Proceedings of the IEEE Conference on Computational Intelligence in Games*, pages 193–200, 2013.